

General Overview

Timesharing is a wonderful step in the development of computers. Usually , when you think of the word you envision a giant IBM 370 or a Cotrol Data Corporation 6600 or some other large computer, and , along with size , you see the mountains of software needed to run a timeshar-ing system.

Well , you need not worry about this software gobbling up your dear-ly bought memory (of which we are all in short supply) . The overall memory use is around ~~XXXX~~ 0B0H (in hexadecimal) or around 180 decimal bytes , plus whatever is needed for two stack areas (which gives a clue on how this prog ram works) . Of course , any revisions which you may make may add to those figures , but there is a long way to go to reach 1K.

Basically , the routine accomplishes timesharing by using a Real Time Clock (RTC) to inte_rrupt the processor every 1/60th of a second and then it switches the stack of one job to the other job's stack area , ending by then executing at the proper place of the second job .

To run the system , just load in the two jobs and the timeshare pro-gram . Then run the timeshate program , which will automatically setup both stacks , and the Input / Output assignments , the start location of both jobs , and other housekeeping chores. It's as simple as this .

or 10 bytes must be allocated . Then we setup into that last 16 bit location the start address of job 2 which will be POPped the first time thru to send us to the beginning of job 2.

Then we set up JOB 1's stack pointer in the usual way by loading the stack pointer with ~~0F00~~H . But then we also put the SP into the HL register so we can store the current stack pointer at SP1.

Next , since I am using a POLY-88 ,we need to set up the screen so each job gets one half of it . In my program , the top half of the screen is used by JOB 1 , and the bottom half is for JOB 2 . We also must setup the cursor for each job , so we store at CURS1 and CURS2 the respective home position for each job's screen portion .

Next , we need to initialize the TIMER location of the POLY-88 , because the POLY-88 monitor is in EPROM and the only way I can get out of the clock routine is to make TIMER equal to one less than zero , so the processor will jump to TSR when TIMER is incremented to zero . So, I store two ~~0FFFF~~H into the 4 byte location TIMER .

Then , TSR installs the actual job switching routine's address into the ETC ISR , so when a 60 Hz interrupt occurs , it will jump to TSR everytime . Finally , setup ends by enabling interrupts and then jumping to JOB 1 .

Now , the actual job switching routine - it puts the current Stack Pointer into HL . Then we load into the A register a counter called JBSW .

When the rightmost bit of JBSW is a 1 , we will switch from JOB 2 to JOB 1 , and vice versa for a 0 . First JBSW is incremented by one and stored back .

The switch first stores the old stack pointer into it's respective storage area , and then reinstates the new job's Stack Pointer . Then it sets up the screen and restores the new job's current cursor position after saving the old job's cursor . Then I reinitialized TIMER to zero minus one so that we will jump to TSR after every RTC clock interrupt . Next we exit via IORET which POPs all the registers in the standard sequences , enables interrupts , and then returns to the new job , thus ending the job switch .

In general , the TSR RTC ISR works by first doing a standard push sequence (part of the POLY-88 monitor) , switches the job's stack pointers , changes the screen parameters and the input addresses , and then restores the new job's registers by jumping to IORET .

Conclusion

The TSR is a simple to use method of time sharing on a microcomputer and is easily implemented on nearly any 8080 machine .

The only changes which may be needed would be patches to your monitor for I/O routines ,and replacements for IORET and the POP sequence of an interrupt service routine .

Hope you enjoy this program , which follows in the appendix .

Appendix

LABEL	OP CODE	OPERAND	COMMENTS
START	CALL	CLEAR	clears screen
LXI	LXI	H,SRA3	SRA3 - keyboard 2 interrupt vector
	SHLD	KSR2	Install kybrd 2 interrupt service routine
	CALL	JBSTG	output query
	MVI	A,'1'	JOB 1
	CALL	JB?	Query end
	SHLD	JB1SA	Store JOB 1's start address
	CALL	CROUT	output a carriage return
	CALL	JBSTG	output query
	MVI	A,'2'	JOB 2
	CALL	JB?	end query
	SHLD	JB2SA	Store JOB 2's start address
	CALL	CLEAR	clear screen ready for jobs
	JMP	BEGIN	goto setup routine
STRG?	DS	'Start. Address of 04'	
STRGJ	DS	' ? ' 04	
KBUF2	DS	2	kybd 2 ISR buffer
JB1SA	DS	2	
JB2SA	DS	2	
KI2	PUSH	H	kybrd input like WHI
	LXI	H,KBUF2	
	JMP	KI1	
KSR2	IN	XXX ADDR	input kybrd data
	LXI	H,KBUF2	
	JMP	IOPUT	
CHR	CALL	WHI	
	INX	H	
STRG	CPI	04	start of string output routine
	JNZ	CHR	
	RET		
JBSTG	LXI	H,STRGJ	
	JMP	STRG	
JB?	CALL	WHI	
	LXI	H,STRG?	
	CALL	STRG	
	JMP	HEXC	
BEGIN	DI		DISABLE INTERRUPTS
	LXI	H,0FF6H	
	SHLD	SP2	setup job 2's stack
	SHLD	JB2SA	
	SHLD	0FF6H	
	LXI	SP,0FF0H	setup job 1's stack
	MVI	A,0F8H/H	
	STAD	SCRHM	
	MVI	A,0FAH	
	STA	SCEND	
	LXI	H,0FAH/H	set up job 2's screen cursor
	SHLD	CURS2	

LABEL	OP CODE	OPERAND	COMMENTS
3 LXI	H, FFFFH		initialize TIMER
	SHLD	TIMER+2	
	SHLB	TIMER	
	LXI	H, TSR	install job switcher at WAKEUP
	SHLD	WAKEUP	
	EI		enable interrupts
	LHLD	JB1SA	
	PCHL		goto job 1
SP1	DS	2	job 1's stack pointer
SP2	DS	2	job 2's stack pointer
CURS1	DS	2	job 1's cursor
CURS2	DS	2	job 2's cursor
JBSW	DB	0	job flip flop
TSR	LXI	H, 0	job switcher
	DAD	SP	
	LDA	JBSW	
	INR	A	
	STA	JBSW	
	RAR		rotate right to put LSB into Carry
	JNC	SWT01	if carry jump to switch to job 1
SWT02	SHLD	SP1	
	LHLD	SP2	
	SPHL		
	MVI	A, 0FAH	screen top
	STA	SCRHM	
	MVI	A, 0FCH	screen bottom
	STA	SCEND	
	LHLD	POS	
	SHLD	CURS1	store old cursor
	; install here keyboard switcher here		
	LHLD	CURS2	reinstate new cursor
	JMP	END	
SWT01	SHLD	SP2	save old stack
	LHLD	SP1	
	SPHL		reinstate new stack
	MVI	A, 0FAH	screen top
	STA	SCRHM	
	MVI	A, 0FCH	Screen bottom
	STA	SCEND	
	LHLD	POS	
	SHLD	CURS2	STORE old stack
	; install here keyboard switcher		
	LHLD	CURS1	
END	SHLD	POS	store new cursor
	LXI	H, FFFFH	
	SHLD	TIMER 2	
	SHLD	TIMER	make timer 1 less than 0
	JMP	IORET	POP all, enable int., and RET to other job
	END		

Equipment Needed

Any 8080 based microcomputer

At least one thousand bytes of Random Access Memory

Vectored interrupts

An input device , hopefully an ASCII keyboard

An output device (a Video Board type screen output works well)

A Fertile imagination to dream up new applications for this program

Vocabulary

K - 1024 bytes.

home - the top leftmost cursor position of a video system's screen.

RTC - real time clock which interrupts the processor every $1 / 60$ th.
of a second by way of the 60 Hertz AC line.

ISR - Interrupt service routine . A routine which increments a counter or
some other small process after an interrupt .

job - a program switched by the processor in a timesharing processor.

Bibliography

Books

Findley , Robert . SCELBI '8080 Software Gourmet Guide and Cookbook'.
Milford , CT . : Scelbi Computer Consulting Inc . , 1976

Manual

POLY-88 Microcomputer Users Manual . Santa Barbara , CA . : Polymorphic
Systems , 1976

Periodicals

Sneed , James R . "Adding an Interrupt Driven Real Time Clock", BYTE
(Nov . 1977) , 2#11:72-74

La Dage , Dan "Interrupts Exposed", Kilobaud (April , 1977) , 2#18-21

La Dage , Dan "Interrupts Exposed - Part 2" , Kilobaud (May 1977) 5:78-80

Gable , G . H . "Using Interrupts To Speed Up An ELM" , BYTE (Jan . 1977) 2:106

Wilcox , Dick "The Hobbyist Operating System - Part 2: Interfacing With The Monitor"
Kilobaud (Feb . 1977) 2:78-79

Krystosek , Paul and McCarthy , John "8080 Programming Notes" BYTE (May 1977)
2#5:136-138

Pamphlets

"The New Breed Of Microcomputers" , Promotional pamphlet published
by Alpha Microsystems , n.d.

"8080 Instruction Set" , promotional pamphlet published by PolymorphicSystems , n.d.

"TEMPOS" , Promotional pamphlet published by Administrative by Administrative
Systems , Inc . , n.d.